Neprekidna isporuka

U okviru ove lekcije govorićemo o pojmu neprekidne isporuke (continuous delivery). Ovo će ujedno biti i jedna od prvih lekcija u kojima se bavimo temom automatizacije procesa izgradnje i integracije aplikacije, pa, nešto kasnije, i automatizacije testiranja.

Pojmovno određenje

Neprekidna isporuka (eng. continuous delivery), često samo CD, predstavlja pristup u softverskom inženjerstvu gde se softver proizvodi u kratkim ciklusima, osiguravajući da se delovi softvera mogu objaviti u bilo kom trenutku. Cilj ovoga pristupa radu jeste da se izrada testiranja i objavljivanje softvera obavljaju većom brzinom i što češće. Sa ovim pristupom kompanije umanjuju troškove i vreme potrebno za realizaciju i smanjuju rizik grešaka, jer se aplikacija spaja iz delova automatski.

Dakle, važno je napomenuti da se ne radi isporuka celokupnog softvera, već se delovi pregledaju, testiraju i debaguju kako pristižu. Važno je odmah napraviti razliku između tri pojma koja se često pomešaju, a nekada čak i greškom smatraju istim pojmom.

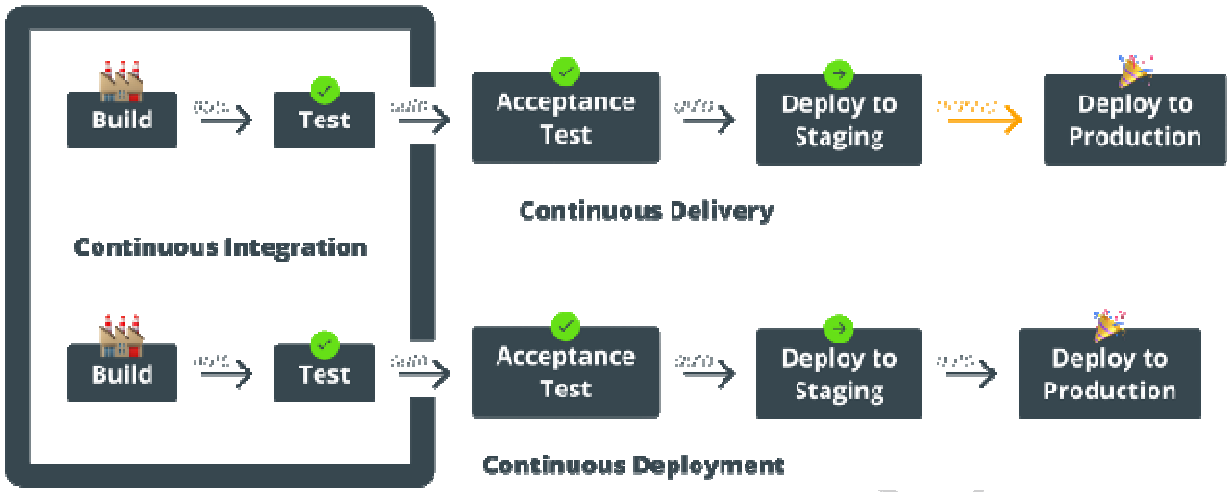
Neprekidna integracija, *neprekida isporuka* i *neprekidno objavljivanje* su tri potpuno različita pojma. Razlog zabune oko njih je u opšte korišćenim skraćenicama za ove oblasti. Tačnije, ovi pojmovi na engleskom su *continuous integration*, *continuous delivery* i *continuous deployment*, sa skraćenicama CI, CD i CD (respektivno); međutim, nekada ćete na sajtovima i u razgovorima naići na CI/CD i CD za ova tri pojma. Ovo može biti zbunjujuće, pa ćemo odmah i jasno definisati ova tri naizgled vrlo slična pojma.

Continuous integration (CI) predstavlja pristup u softverskom inženjerstvu gde programeri vrše integraciju koda u deljeni repozitorijum, gde se više puta u toku jednog dana proverava kod i vrši testiranje integracije. U okviru ovog pristupa vrši se automatsko kreiranje celine ili verzije softvera (build), ali se te verzije ne objavljuju, već se čekaju dalje celine pa se dve celine testiraju i taj proces se ponavlja sve dok se ne dobije završen proizvod.

Continuous delivery (CD) je pristup u kojem programeri razvijaju, verzionišu, testiraju i objavljuju celine programa u veoma kratkim periodima. Naravno, ovo u velikoj meri zavisi od procesa automatizacije u svakoj od faza. Ovaj pristup možemo gledati kao nastavak procesa CI, s tim što se, čim jedna celina prođe testiranje integracije, ona pod kontrolom razvojnog tima tretira kao završena i spremna za objavljivanje. Ovaj pristup se koristi kada se radi update postojećeg softvera, gde je potrebno da se za kratko vreme, a nekada i bez prekida rada programa, izvrši promena na samoj aplikaciji.

I na samom kraju, u pristupu continuous deployment (CD), slično continuous delivery procesu, sav kod se razvija, verzioniše, testira i objavljuje u kratkim periodima, ali je, za razliku od prethodnih, ovaj proces toliko automatizovan da nema ljudske intervencije. Svi testovi se obavljaju automatski kako se fajl okači na repozitorijum i ukoliko prođe testove, automatski se objavljuje. Ovo je najbrži vid rada, ali iziskuje ekstremno precizne testove.

Pogledajmo uporedan prikaz ovih procesa:



Slika 7.1. Uporedni prikaz pristupa neprekidnom razvoju[1]

Kao što možete videti, neprekidna integracija se u svakom slučaju obavlja automatski; nakon deljenja koda na nekom zajedničkom okruženju ili repozitorijumu, izvršavaju se testovi integracije koji su prethodno napisani. Ako kod prođe tu fazu, označava se kao završen. Kako se nivo povećava, ide se ili na delivery ili na deployment pristup. Ukoliko smo u okviru delivery pristupa, tada se vrše dodatni testovi i ukoliko program prođe i te testove, dolazi u staging fazu, gde je označen kao završen, ali čeka finalno objavljivanje programa, a u deployment pristupu, ceo je proces automatski i prolaskom kroz testove, program direktno ide kod klijenta – dakle, bilo u integraciju u postojeći sistem koji klijent koristi, na server i slično.

Kako smo se upoznali sa testovima integracije i njihovom logikom, ovde ćemo se fokusirati na continuous delivery. Videćemo prethodno spomenutu integraciju, ali i kako izgleda sam delivery proces.

Pre nego što nastavimo dalje, ostalo nam je još par pojmova koje moramo imati u svom rečniku da bismo razumeli procese automatizacije i ujedno korišćenje programa u ovoj oblasti.

Prvi pojam je *build*. Ukoliko se do sada niste sreli sa ovim pojmom, build ukratko predstavlja verziju programa, često verziju programa koji još nije završen. Dakle, ako kažemo da je naš program build verzije 1.0, to predstavlja da je to prva verzija softvera koji počinje da predstavlja celinu, ali na kojem je ostalo još razvoja. Svaka nova celina programa koja se npr. testira na integraciju će predstavljati jednu verziju, jedan build. I stoga možete videti verzije poput 1.01, 2.31 i slično.

Za razliku od build verzije, imamo i release verziju. Release verzija predstavlja verziju programa koja je stabilna i koji upotrebljavaju korisnici. Ove verzije programa možemo videti na svim programima našeg računara, pa npr. Google Chrome trenutno ima release verziju 89.0.4389.114.

I za kraj, pojam *deploy* u kontekstu software deploymenta predstavlja celinu koda ili celu aplikaciju koja je postavljena na neko okruženje, gde se testira ili od strane razvojnog tima, kroz testove integracije ili druge vidove testova, ili alternativno od strane male grupe korisnika, gde se testira rad programa u realnim uslovima.

[1] https://medium.com/swlh/ci-vs-cd-vs-cd-e102c6dd88eb

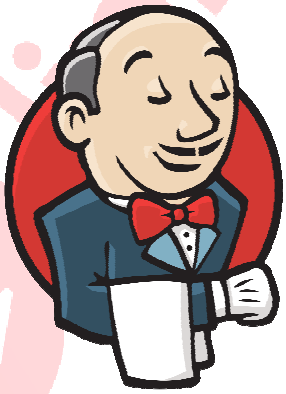
Sada smo se upoznali sa nekim pojmovima koji će nam biti potrebni u daljem radu. Ono što želimo da napomenemo jeste da se ovi procesi automatizacije sprovode u većim organizacijama na velikim projektima, gde su potrebni značajno vreme i troškovi za realizaciju projekata. Kao takvi, dakle, nisu deo svakodnevnice i procesom automatizacije i kontrole okruženja često upravljaju vođe timova, project menadžeri i slični kadrovi u organizaciji. Sada je važno upoznati se sa procesima, jer već u daljem razvoju mogu postati deo vaše svakodnevnice, naročito u radu sa Python jezikom, jer je on jedan od najpopularnijih u oblasti testiranja.

Jenkins

Verujemo da se sada pitate – ali kako se ta automatizacija postiže? Spominjali smo da svaki od pristupa uključuje neki repozitorijum/okruženje gde programeri dostavljaju svoje delove koda, koji se testiraju i po automatizmu pomeraju u dalje faze.

Postoji mnogo varijacija kako izvoditi neprekidnu isporuku. Ovde ćemo prikazati jedan vrlo popularn program za automatizaciju: Jenkins. Jenkins se koristi ne samo za Python, već i za Java, PHP, JavaScript i druge jezike koji uzimaju značajan deo tržišta softvera.

Krenimo od toga šta je to Jenkins. Jenkins je vodeći open source server za automatizaciju procesa razvoja softvera. Njega odlikuju velika zajednica programera i stotine plugina koji olakšavaju rad i on omogućava upravo neprekidnu isporuku i automatizaciju testova. Velike su šanse da u kasnijem radu u nekom trenutku dođete u dodir sa organizacijom koja koristi Jenkins za svoju automatizaciju razvoja.



Slika 7.2. Jenkins logo[2]

Kako je Jenkins server, dakle serverska aplikacija, on se postavlja na neki vid web servera. Ovo omogućava programu da ostvaruje potrebne konekcije sa drugim udaljenim računarima, serverima za skladištenje podataka i slično. Upravo tu primenu ćete videti u okviru većine kompanija, gde postoji odvojen server na kojem se nalazi Jenkins i vi kao član tima dobijate link do njega.

[2] https://commons.wikimedia.org/wiki/File:Jenkins_logo.svg

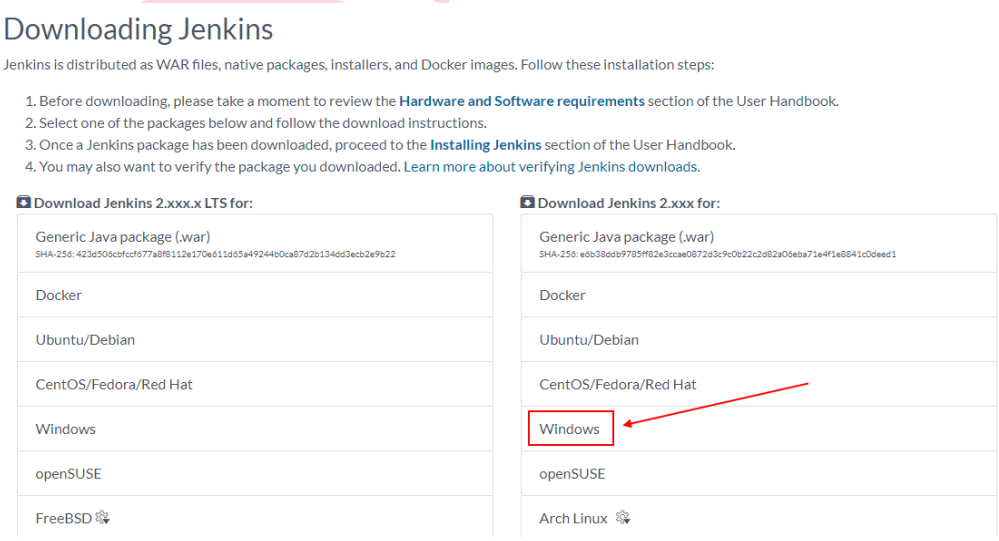
Naravno, mi želimo da radimo sa Jenkinsom, ali nismo deo velikog razvojnog tima. Srećom, to nam je olakšano i možemo preuzeti instalaciju Jenkinsa za naš operativni sistem, gde će on na našem računaru kreirati jedan virtuelni server i na taj način nam je omogućeno da podesimo ovaj program i upoznamo se sa njim.

Instalacija Jenkins servera

Prvi korak instalacije je, naravno, da preuzmemo potreban fajl. Na sledećem linku: https://www.jenkins.io/ preuzimamo samu instalaciju, ali pored toga, ovo je i zvanična stranica programa, pa u okviru nje možemo videti plugine, dokumentaciju, blog, forume i slično. Ali naravno, nas trenutno zanima instalacija, pa na narednoj slici možete videti link za pristup download sekciji:

Slika 7.3. Zvanični sajt Jenkins.io i opcija za preuzimanje programa

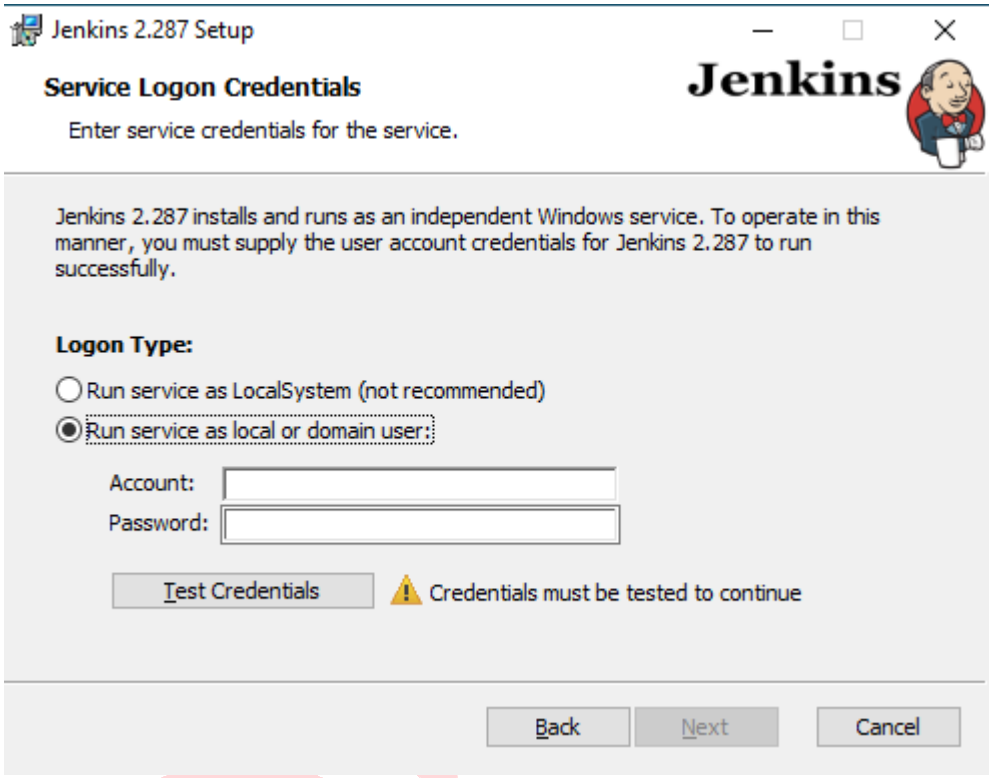
U okviru sledeće stranice sajta, pronalazimo sekciju Downloading Jenkins i u koloni sa desne strane, gde se nalaze najnovije verzije programa, biramo verziju za naš operativni sistem. Konkretno u našem slučaju, koristimo Windows i stoga biramo tu verziju programa.



Slika 7.4. Preuzimanje Windows verzije programa

Nakon izbora opcije, automatski se započinje preuzimanje instalacionog .msi fajla. Kada je fajl preuzet na računaru, možete pokrenuti i instalaciju programa. Sami koraci instalacije su jednostavni, nema potrebe za menjanjem nekih podešavanja, osim podešavanja navedenih u nastavku.

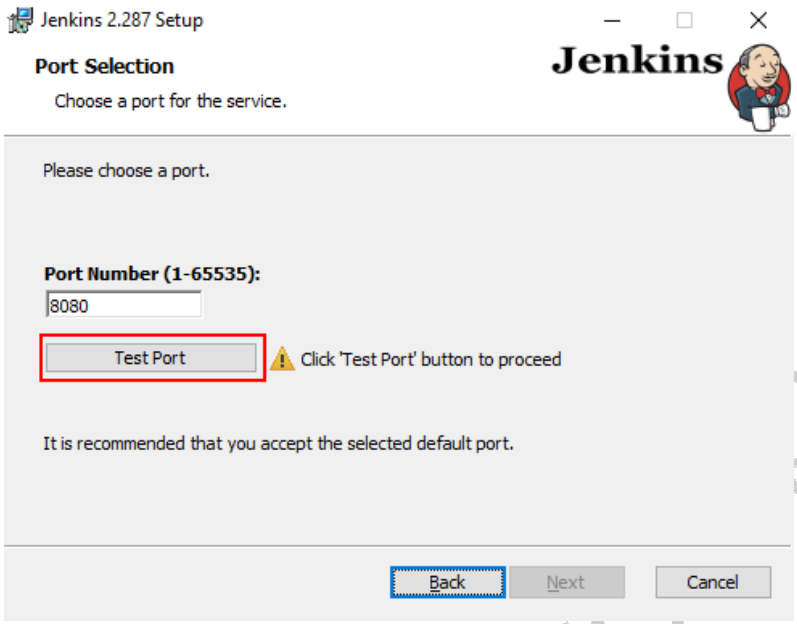
Na trećem koraku instalacije, može doći do odstupanja ukoliko niste administrator računara; tada će se pojaviti prozor kao na slici u nastavku:



Slika 7.5. Login u administratorski nalog operativnog sistema

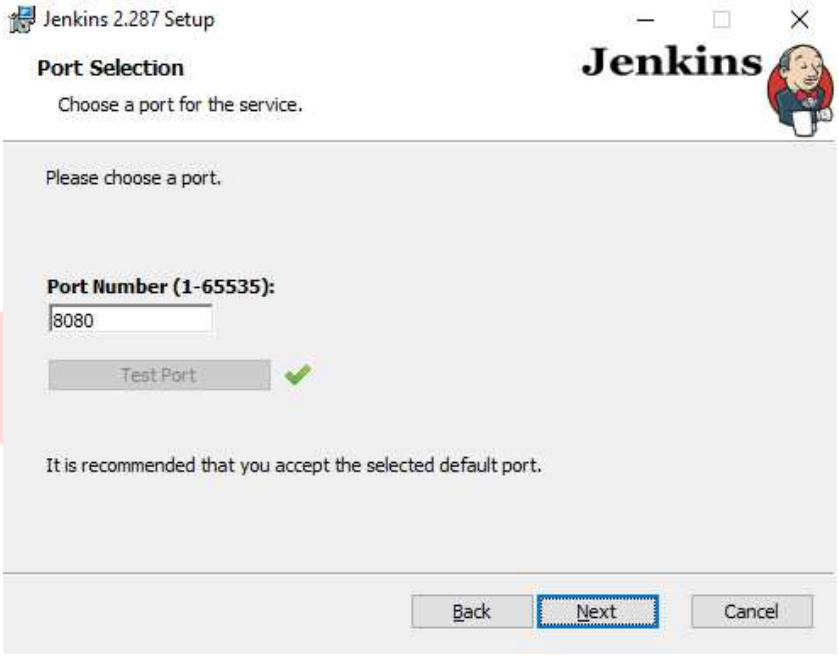
U slučaju pojave ovog prozora, potrebno je u polja uneti login podatke administratorskog naloga. Ukoliko ste jedini korisnik, onda je dovoljno odabrati opciju *Run service as LocalSystem* i nastaviti dalje.

Sledeći korak je provera dostupnog porta. Jenkins kao server mora imati jedan dostupan port koji je otvoren za njegovu komunikaciju. Podrazumevani port je 8080 i bilo bi poželjno koristiti taj port. U slučaju da to nije moguće, otvorena je mogućnost promene broja porta. Da biste nastavili dalje, a ujedno i proverili da li je port 8080 dostupan, potrebno je odabrati taster *Test Port* (slika 7.6).



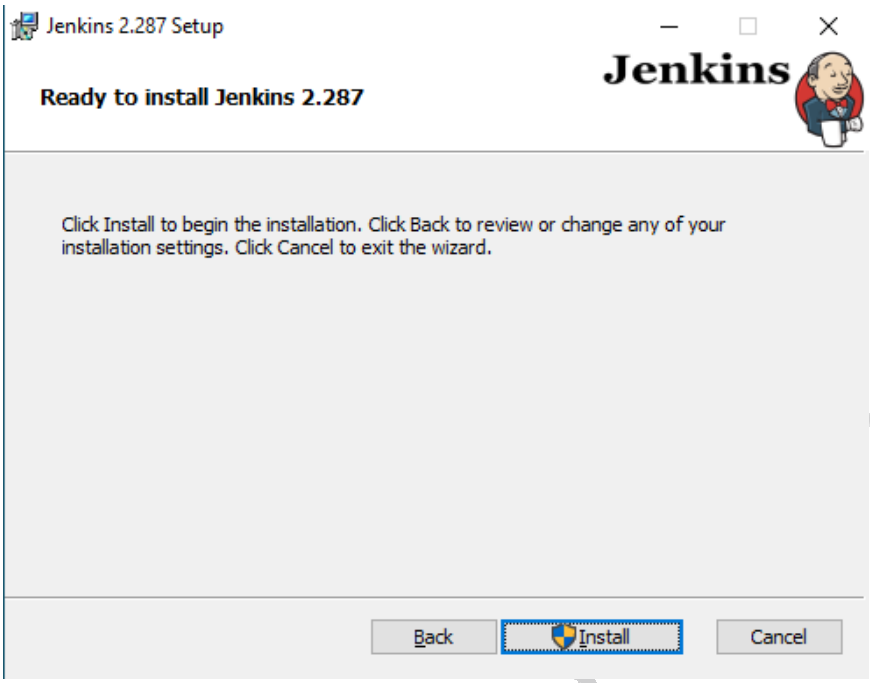
Slika 7.6. Testiranje dostupnog porta za Jenkins

Nakon uspešno završenog testa, pojaviće se taster *Next*.



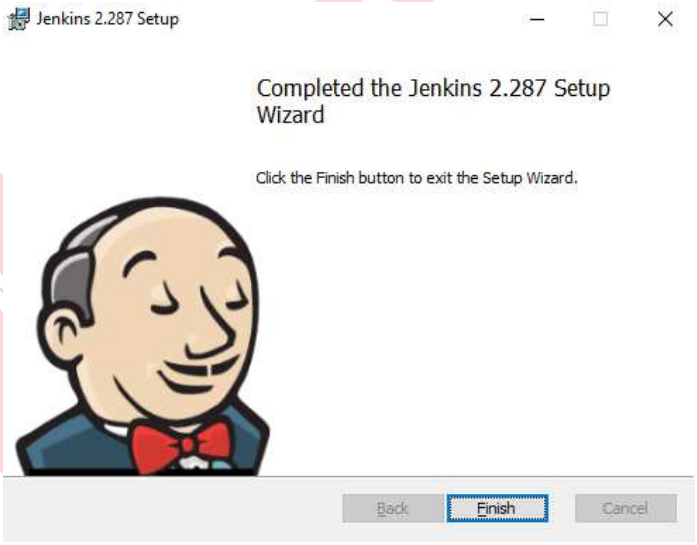
Slika 7.7. Uspešno završena provera porta 8080

Naredne korake instalacije nije potrebno menjati – dakle, dovoljno je da vršite izbor tastera *Next* dok ne dođete do prozora za potvrdu instalacije, gde će i započeti samo kopiranje fajlova programa na vaš računar.



Slika 7.8. Potvrda instalacije programa

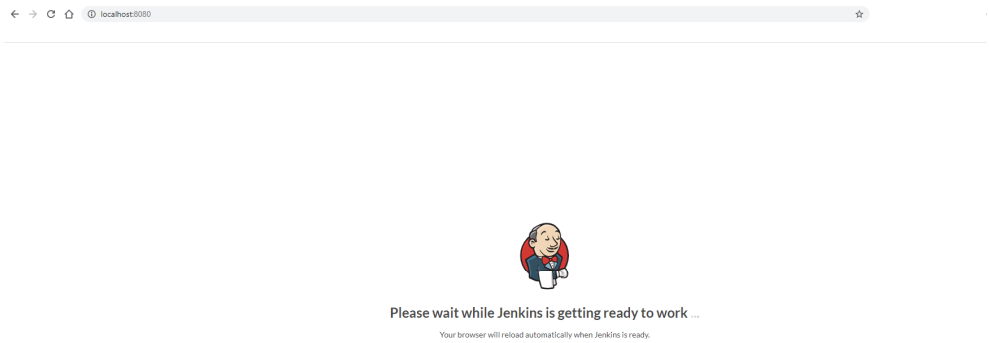
Kada se uspešno završi proces kopiranja fajlova, pojaviće se sledeći prozor:



Slika 7.9. Uspešno završena instalacija programa

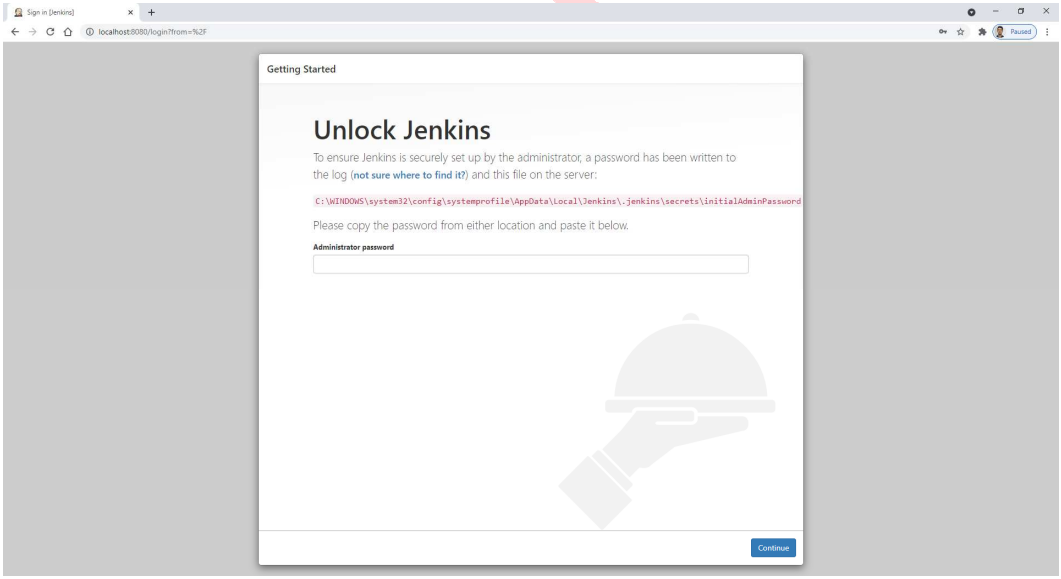
Nakon izbora *Finish* tastera, završena je instalacija programa, ali nije i njegovo podešavanje. Automatski će se otvoriti prozor vašeg podrazumevanog pregledača, koji će pristupiti adresi localhost:8080. Dakle, Jenkins će pristupiti vašoj adresi računara i portu 8080. Ovo je važno da zapamtite, jer ćete svaki naredni pristup interfejsu Jenkinsa obavljati putem ove adrese, koju kucate u adresnoj liniji pregledača.

Sada pristupamo podešavanju samog programa i automatski otvoren prozor pregledača bi trebao da izgleda slično slici u nastavku:



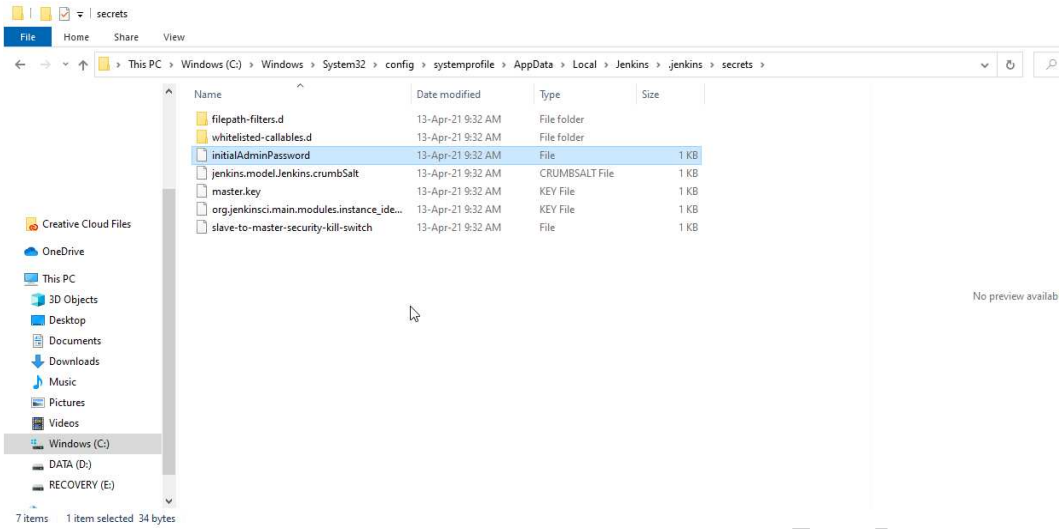
Slika 7.10. Podešavanje Jenkinsa unutar pregledača

Nakon kratkog vremena dolazi se do prvog koraka, a to je unos lozinke kojom potvrđujemo administratorske privilegije.



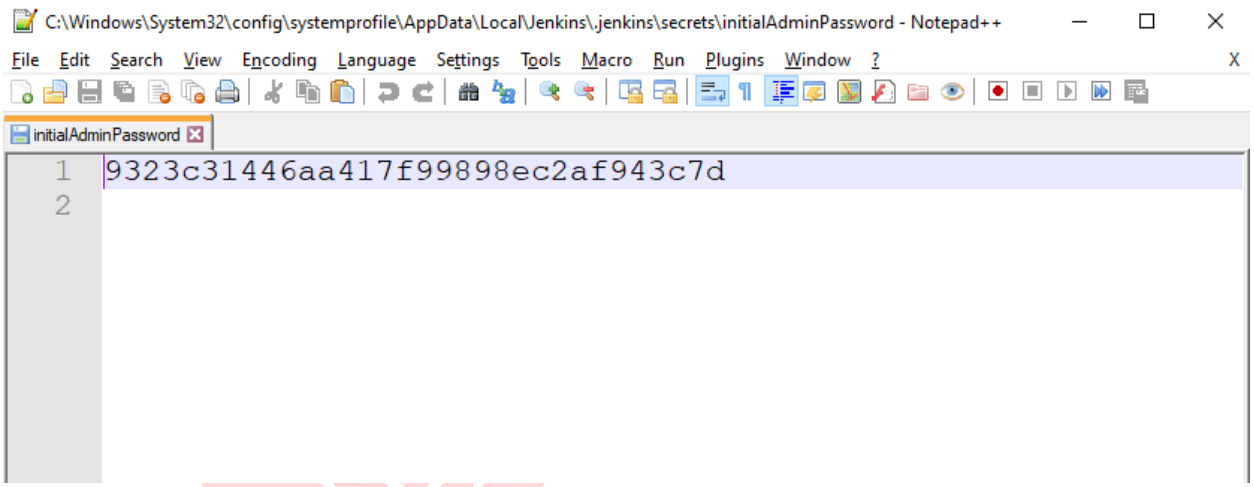
Slika 7.11. Dijalog za unos lozinke

U okviru same notifikacije možete videti putanju do fajla koji sadrži ovu lozinku. Obratite pažnju na to da lozinka važi samo za jedno otvaranje. Najlakše nam je da markiramo ovu putanju i kopiramo je u Windows Explorer. U trenutku kada kopirate link, otvoriće se *Open With* dijalog, gde će vaš operativni sistem pitati kojim programom želite da otvorite fajl. Izaberite bilo koji tekst editor – Notepad, Notepad++, Visual Studio Code itd.



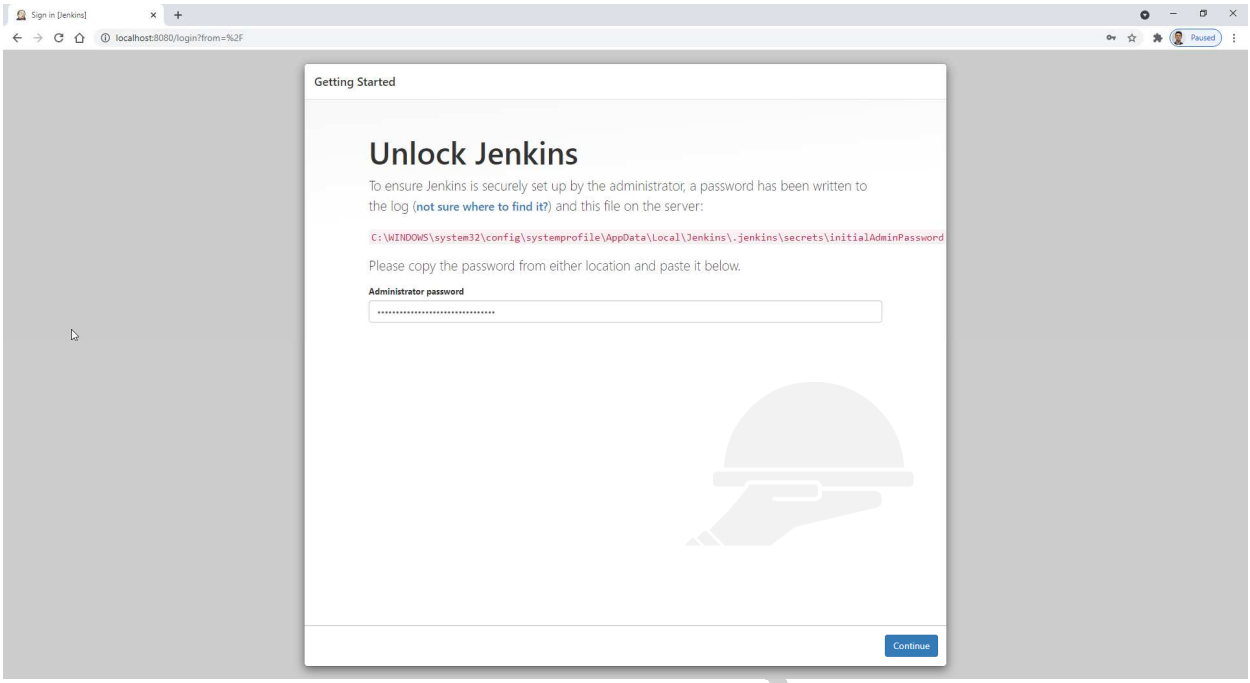
Slika 7.12. initialAdminPassword fajl

Kada otvorite fajl pomoću tekst editora, u okviru njega ćete pronaći samo jednu liniju i u okviru nje lozinku, poput prikazane na slici:



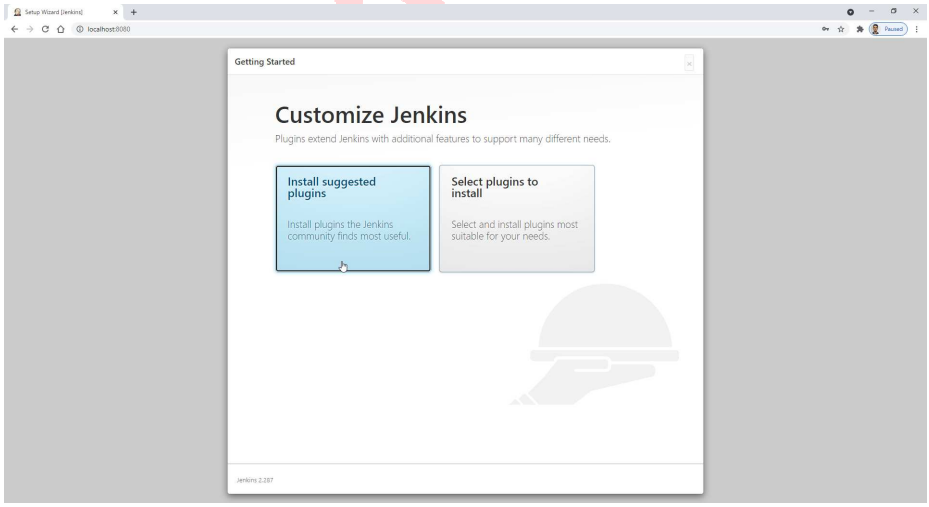
Slika 7.13. Loznika unutar initialAdminPassword fajla

Sada je dovoljno iskopirati lozinku u pregledač i polje za unos lozinke i nakon toga odabrati taster *Continue*.



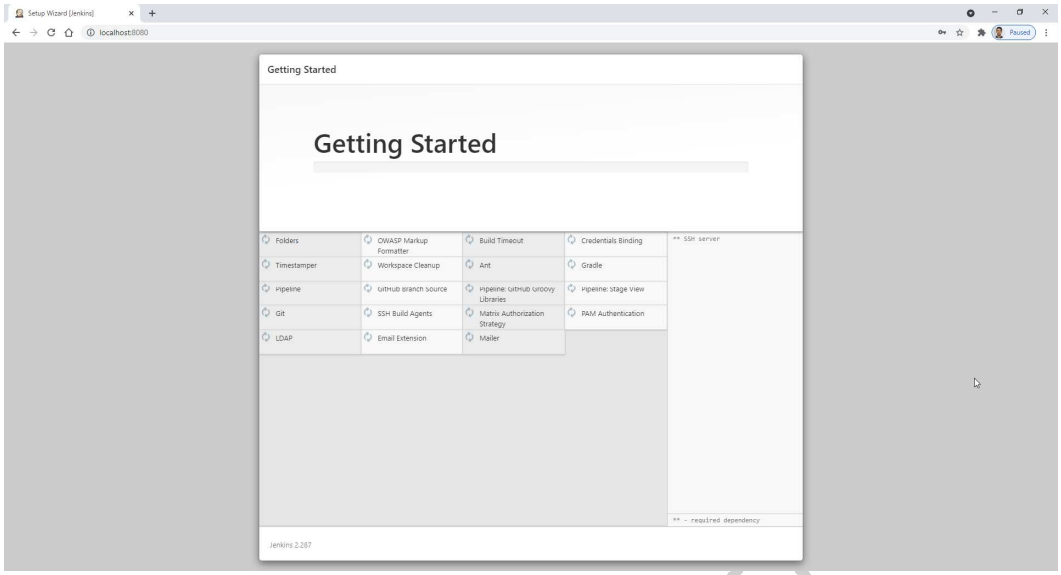
Slika 7.14. Unos Jenkins admin lozinke unutar pregledača

Na sledećem prozoru dolazimo do instalacije plugina. Dodaci su veliki deo Jenkinsa; oni često koristi plugine za koje ni ne znamo da su aktivni. Preporučen je izbor opcije *Install suggested plugins*, gde dobijamo automatsku instalaciju osnovnih plugina za rad sa Jenkinsom.



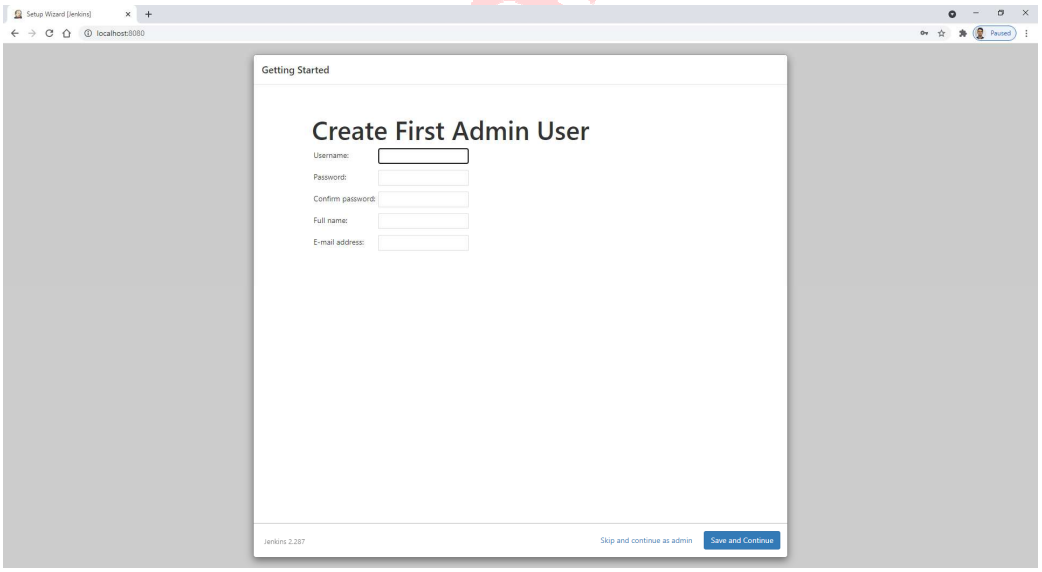
Slika 7.15. Automatska instalacija plugina

Sledeći prozor će prikazivati instalaciju pojedinačnih plugina; dovoljno je samo sačekati završetak instalacije i odabrati taster *Continue*.



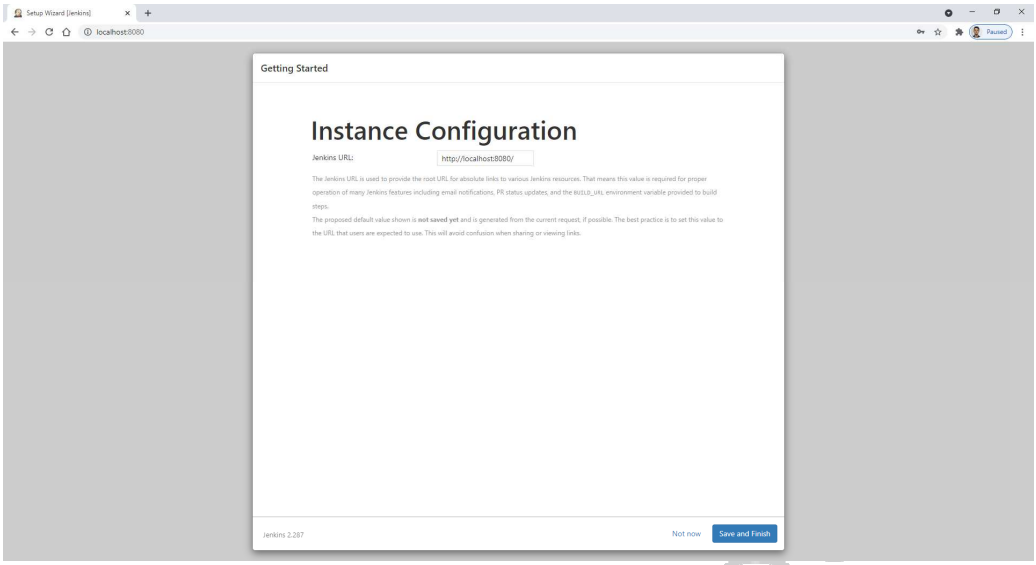
Slika 7.16. Instalacija plugina

Na sledećem prozoru dolazimo do registracije korisnika. Ovaj korak se može preskočiti i u tom slučaju Jenkins će iskoristiti podatke vašeg korisničkog naloga operativnog sistema, ali je ipak bolje registrovati nalog sa vašim imenom i prezimenom, kao i email adresom. Najbolje je već sada pratiti postupke kako bi bilo očekivano u jednoj organizaciji, a u tom slučaju bi se očekivala realna registracija korisnika; stoga ćemo i mi uraditi tako.



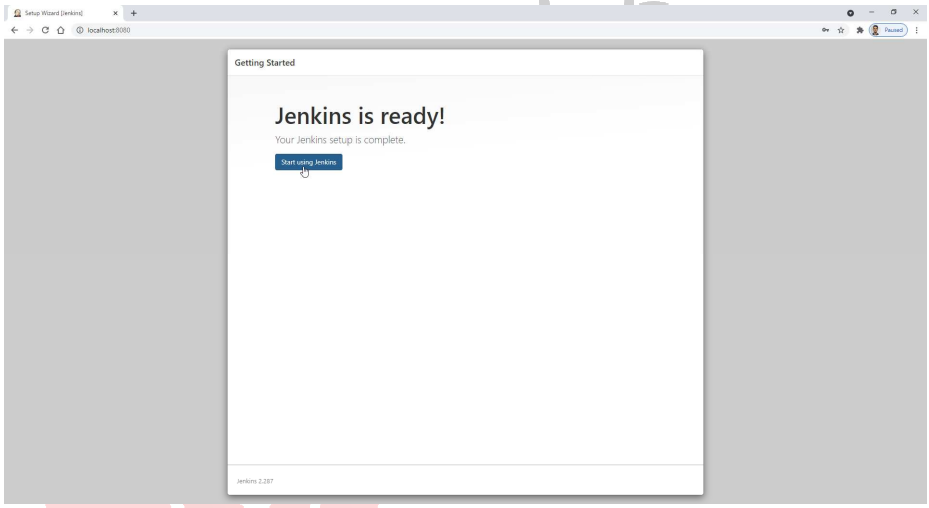
Slika 7.17. Podešavanje administratorskog naloga

Sledeći korak je generisanje linka za pristup Jenkinsu. Kako mi radimo na svojim računarima, naš link ostaje localhost sa portom 8080. U slučaju podešavanja Jenkinsa na serveru, u ovom delu bismo definisali na kojem domenu se nalazi i kako će link izgledati, jer će se taj link deliti sa drugim zaposlenima na projektu.



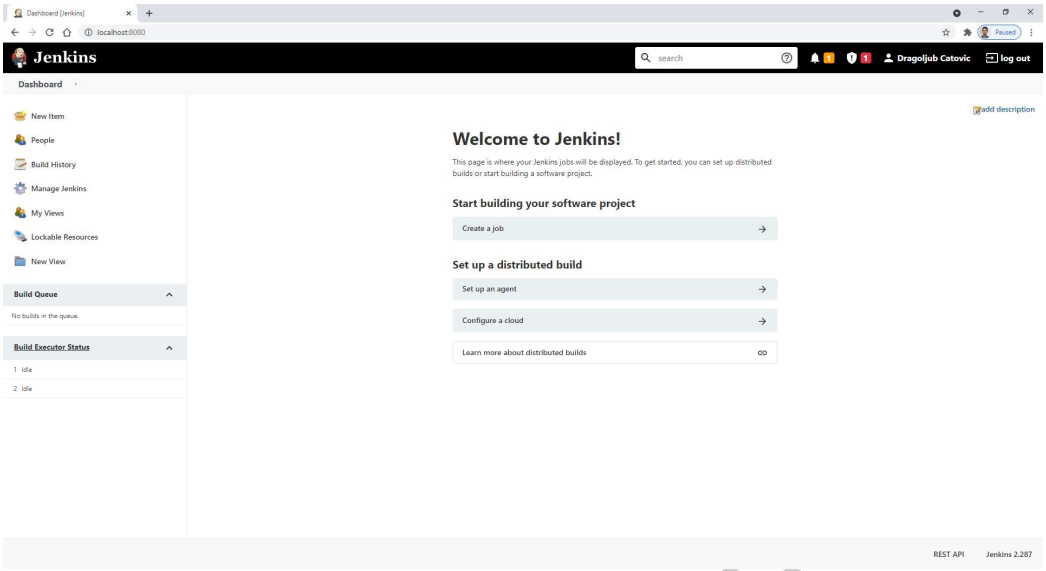
Slika 7.18. Podešavanje adrese Jenkins servera

Došli smo do finalnog koraka, a to je potvrda instalacije i pristup glavnoj stranici Jenkinsa.



Slika 7.19. Potvrda o uspešnom podešavanju Jenkinsa

Na slici možete videti kako izgleda glavni prozor programa:



Slika 7.20. Glavni prozor / Dashboard Jenkinsa

Git instalacija

Pre nego što nastavimo sa radom, potreban nam je još jedan program. Kako je Jenkins instaliran na našem računaru bez pristupa udaljenim računarima, ne možemo vršiti razmenu podataka na mreži. Ali, naravno, postoji način kako se to postiže i to je ujedno jedan od najpopularnijih metoda: povezivanje Jenkinsa putem Gita sa GitHub repozitorijumom. Na ovaj način pristupate repozitorijumima koje možete da kreirate sami ili ih je kreirao čitav razvojni tim. Na taj način Jenkins dobija podatke koje dalje obrađuje i testira, ali pre toga moramo imati instaliran Git na računaru. S obzirom na to da smo i ranije koristili Git, možete prvo proveriti da li vam je već instaliran. To možete proveriti pokretanjem terminala operativnog sistema i unosom komande:

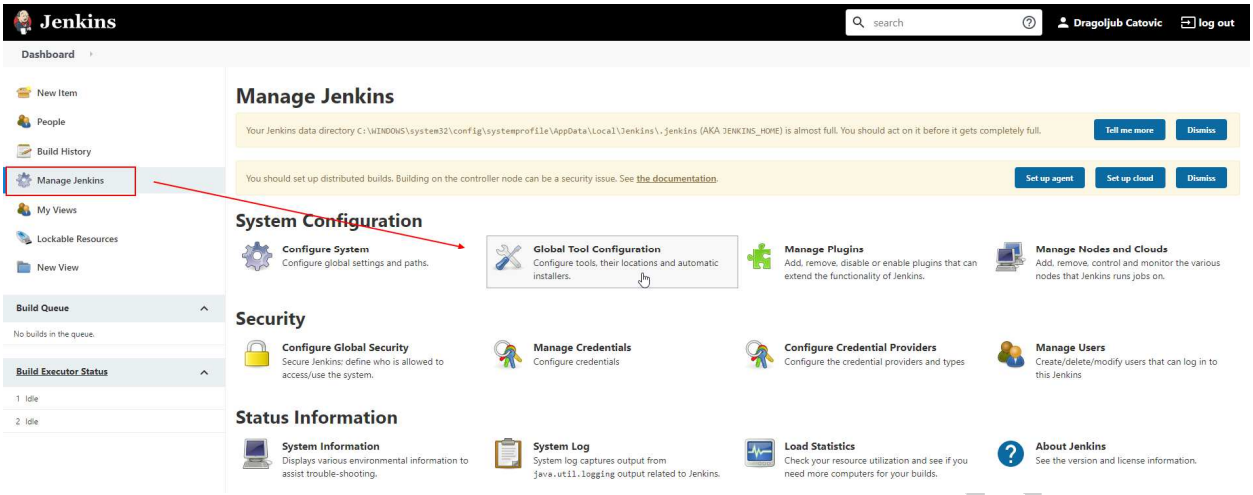
```
git --version
```

Nakon ove naredbe, prikazuje se tačna verzija Gita i tip operativnog sistema. U slučaju greške, poput „git command not recognized”, Git nije instaliran na vašem računaru.

Ukoliko Git nije instaliran, možete ga preuzeti sa sledećeg linka: https://git-scm.com/downloads

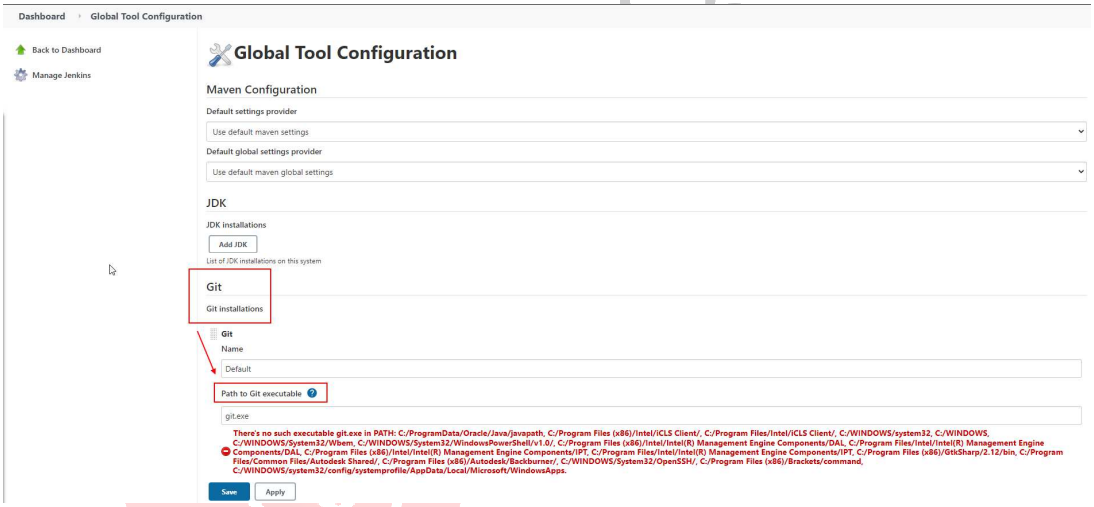
Koraci instalacije su jednostavni i nema potrebe za menjanjem podrazumevanih podešavanja Gita. Nakon uspešne instalacije, možemo nastaviti sa korišćenjem Jenkinsa i kreiranjem našeg prvog projekta koji će biti povezan sa GitHub repozitorijumom.

Prvi korak pri kreiranju Jenkins projekta koji obavlja integraciju kroz GitHub jeste da povežemo našu instalaciju Gita sa Jenkinsom. Stoga je potrebno pristupiti opciji *Manage Jenkins* i sekciji *Global Tool Configuration*.



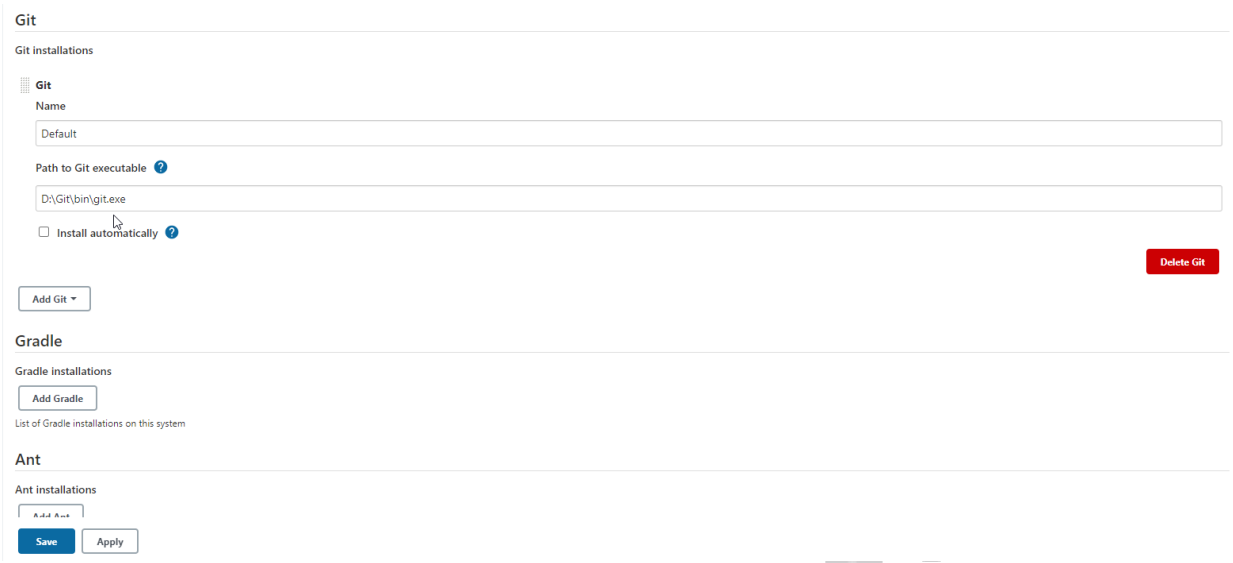
Slika 7.21 Pristup podešavanjima alata

Unutar sekcije *Global Tool Configuration* potrebno je pronaći sekciju *Git* i u zavisnosti od podešavanja računara, Jenkins automatski može potražiti Git.exe fajl i ukoliko ga ne pronađe, videćete niz grešaka kao što je prikazano na slici u nastavku



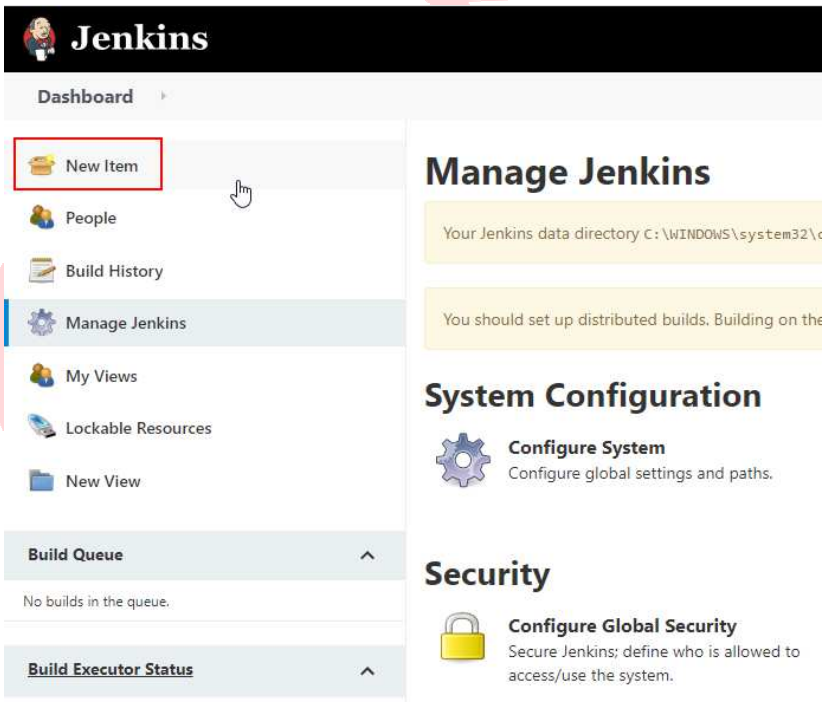
Slika 7.22 Pristup podešavanjima Gita

Ovo, naravno, nije razlog za brigu – samo je potrebno u polju *Path to Git executable* naznačiti tačnu putanju gde se nalazi Git instalacija. Pa npr. u našem slučaju instalacija se nalazi na putanji D:\Git\, a exe fajl u folderu bin\git.exe i upravo ovo navodimo unutar polja.



Slika 7.23. Definisanje putanje do Git instalacije

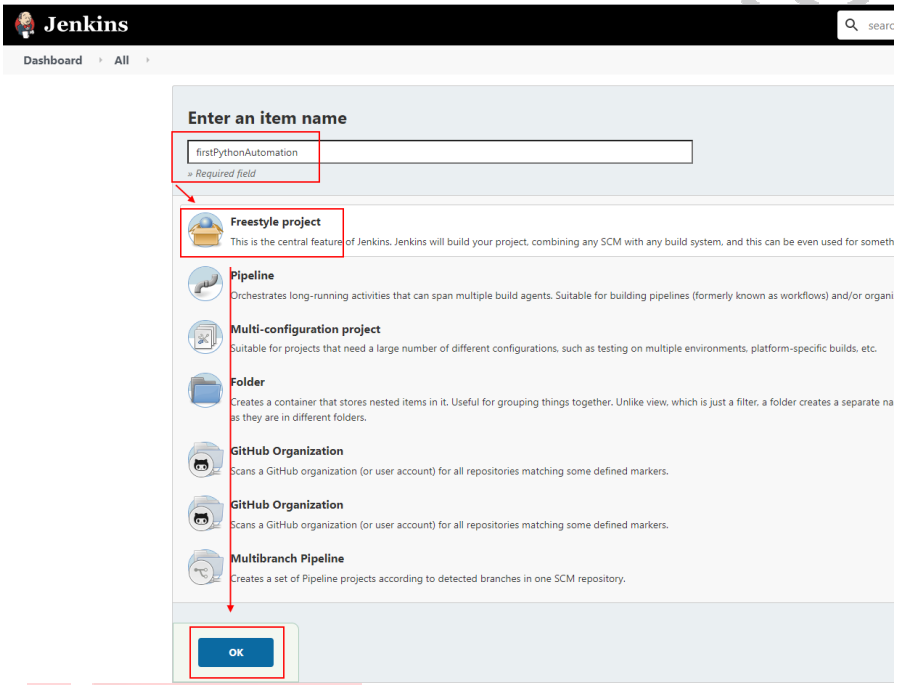
Naravno, u vašem slučaju putanja može biti potpuno drugačija i vi unosite putanje onako kako su organizovane na vašem računaru. Nakon toga potvrđujemo izmenu sa *Apply* i čuvamo izmene sa *Save*. Ovo će nas vratiti na *Manage* sekciju Jenkinsa. I sada možemo kreirati naš prvi projekat. Ovo ćemo uraditi pomoću opcije *New Item*.



Slika 7.24. Kreiranje novog projekta izborom opcije New Item

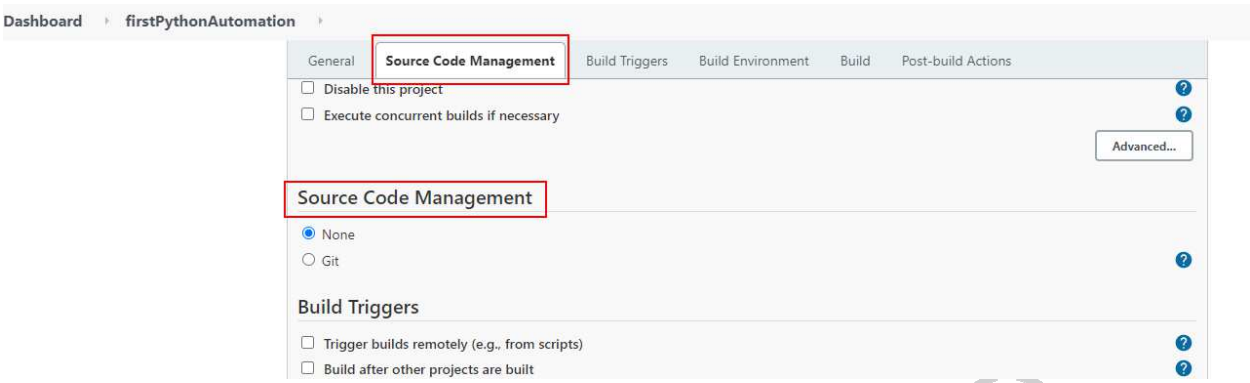
U novom prozoru dolazimo do unosa prvih parametara projekta. Prvenstveno, potrebno je da definišemo naziv projekta. Ovo je, naravno, potpuno proizvoljno; naziv ćemo definisati kao firstPythonAutomation. Sledeći faktor je izbor projekta. Kao što možete videti, nudi se više varijacija. Izbor varijacije zavisi od tipa projekta, kompleksnosti i veličine tima, pa stoga postoje verzije poput *Pipeline* i *Multi-configuration project*. Ove opcije se koriste za velike organizacije, kada se jedan projekat deli u više smerova i gde postoje kompleksnije automatizacije. Nakon toga imamo tri varijacije projekata za GitHub organizacije. Ovo će takođe biti čest tip projekata, jer organizacije koriste plaćene verzije GitHuba da definišu članove svoje organizacije i da poseduju veliki broj privatnih repozitorijuma.

Nama je potrebna opcija *Freestyle project*. Sa ovom opcijom imamo najveću slobodu da sami definišemo korake projekta i podesimo sve parametre kako nama odgovaraju. Stoga biramo tu opciju.



Slika 7.25. Kreiranje novog FreeStyle projekta

Nakon što odaberemo opciju kreiranja projekta, dolazimo do prozora u kojem imamo više kartica, sa velikim brojem opcija. Naravno, većina opcija zavisi direktno od tipa projekta, a veći broj njih i od programskog jezika koji koristimo. Kako mi želimo projekat sa integracijom GitHub repozitorijuma, jedina opcija koja nam je potrebna se nalazi pod *Source Code Management* opcijom.



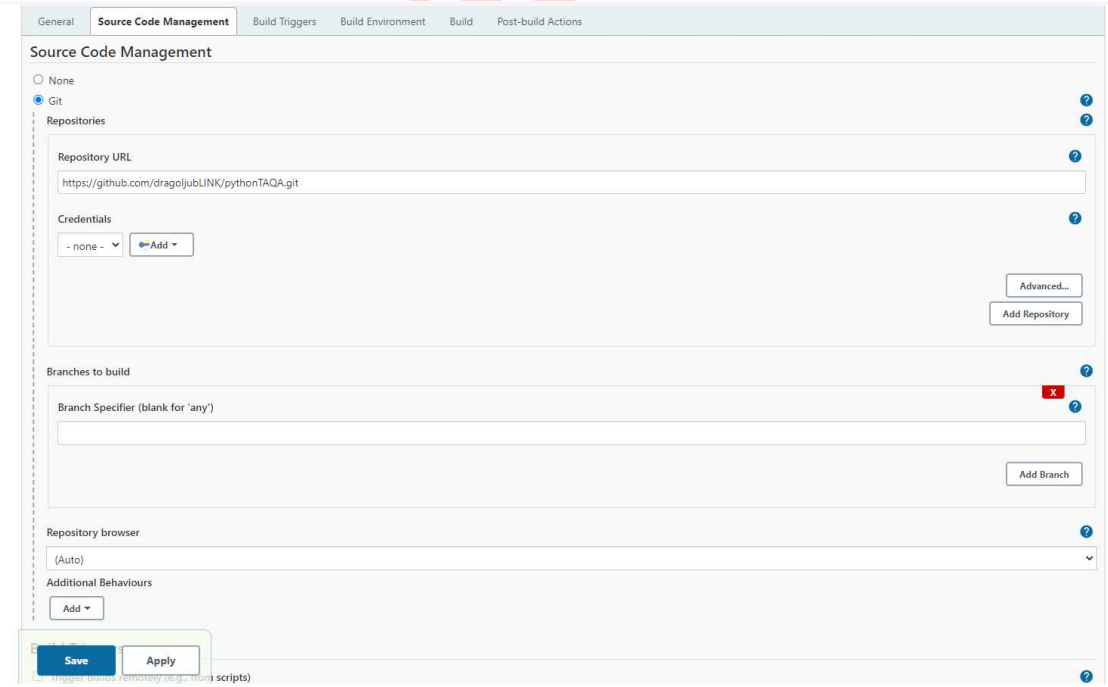
Slika 7.26. Izbor sistema za upravljanje kodom

Opcija *Source Code Management* omogućava nam da izaberemo na koji način dobijamo izvorni kod koji dolazi u Jenkins. Dakle, to je kod koji programeri kreiraju i šalju u celinama na testiranje. Kako mi želimo da kod stiže sa GitHub repozitorijuma, biramo opciju *Git* i u polju *Repository URL* unosimo link do Github naloga, ali obratite pažnju: link mora biti u formi kao da kloniramo repozitorijum.

Za potrebe ovog kursa, kreiran je javni repozitorijum koji će biti korišćen za Jenkins; link je: https://github.com/dragoljubLINK/pythonTAQA.git

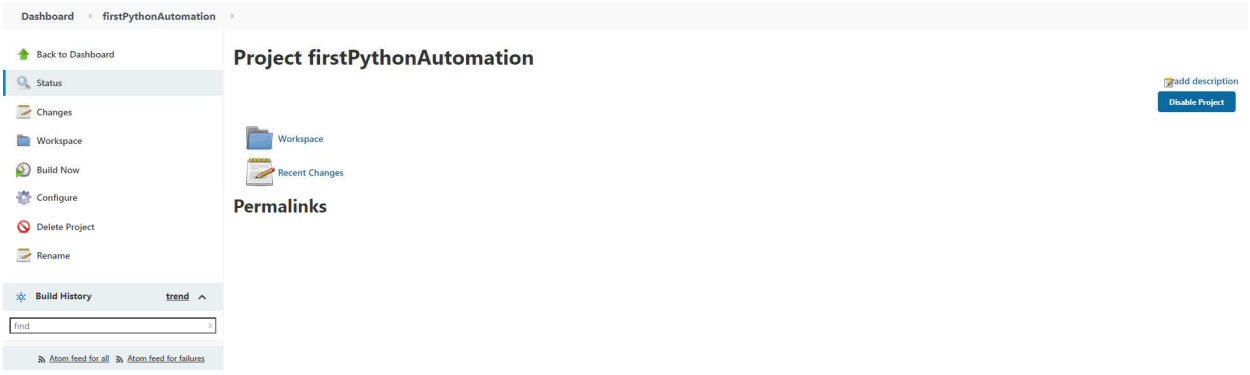
Ukoliko želite da koristite neki drugi, slobodno iskoristite svoj postojeći repozitorijum ili kreirajte novi registracijom na: https://github.com/

U nastavku možete videti definisan link do repozitorijuma unutar Jenkinsa:



Slika 7.27. Parametri podešavanja Git repozitorijuma

Obratite pažnju na to da pod sekcijom *Branches to build* treba ostaviti prazno polje, jer želimo da Jenkins ima pun pristup repozitorijumu. Nakon toga izaberite *Apply* pa *Save*. Kako je korak završen, Jenkins nas vodi na početni prozor našeg projekta.

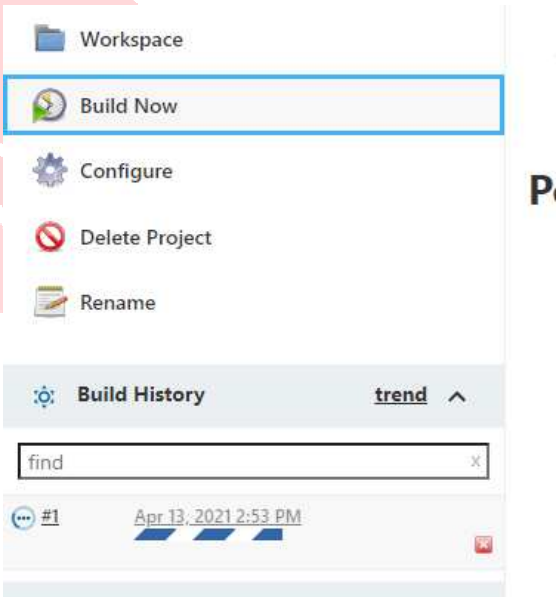


Slika 7.28. Parametri podešavanja Git repozitorijuma

Podrazumevano je otvorena *Status* opcija, koja prikazuje dešavanja u okviru projekta; naravno, kako smo tek kreirali projekat, nemamo neka dešavanja ovde, ali kasnije, kako se obavlja rad na projektu, ovde dobijamo brz pregled svih dešavanja.

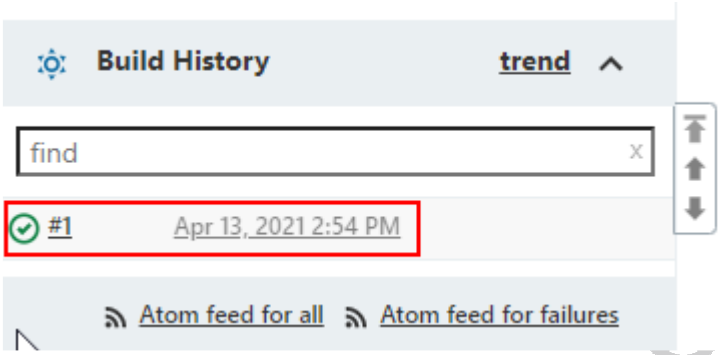
Sledeća stavka je *Changes*; u ovom prozoru dobijamo pregled svih promena na fajlovima projekta. Opcijom *Workspace* pregledamo folder u kojem Jenkins čuva fajlove projekta. Opcijom *Build now* kreiramo build ili, kako smo rekli, prvu verziju našeg programa. *Configure* opcija pruža detaljnija podešavanja za sam projekat i njegov projektni folder, a sa poslednje dve opcije vršimo brisanje projekta i promenu njegovog naziva, respektivno.

Da bismo obavili prvu integraciju i na taj način preneli fajlove na Jenkins server, potrebno je da izaberemo opciju *Build Now*.



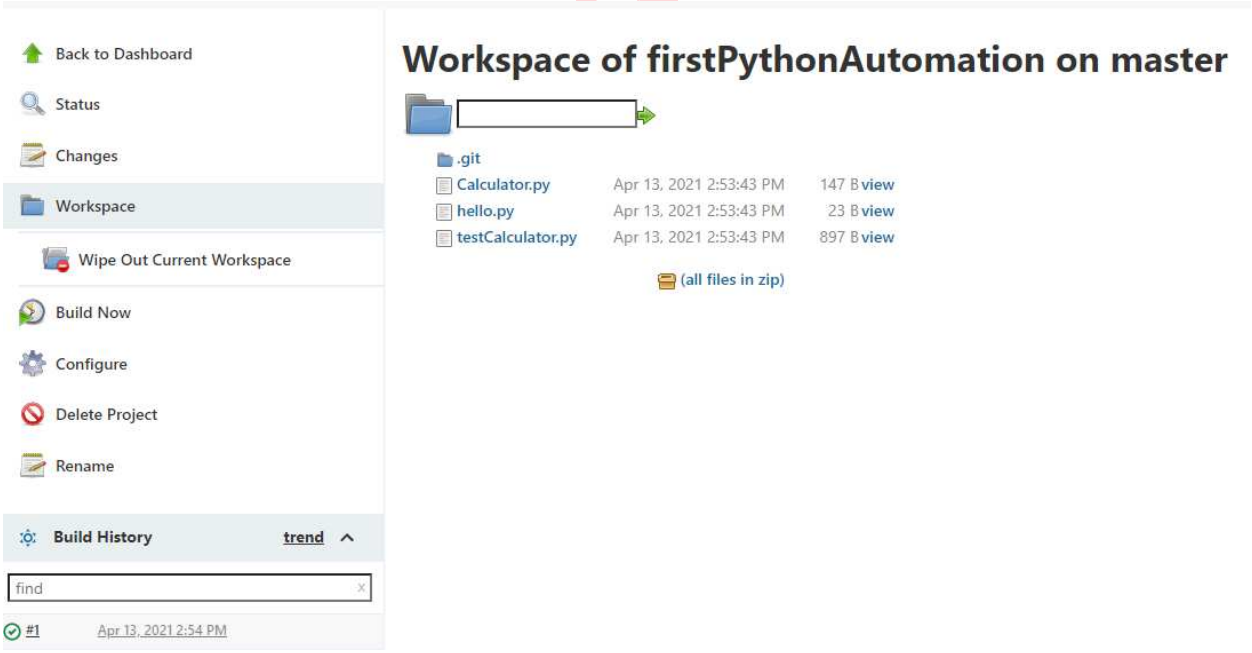
Slika 7.29. Kreiranje prvog builda projekta

Na donjem delu slike 7.29. možete videti da se pojavljuje broj 1 i da je neki vid učitavanja u toku. Jenkins u ovom trenutku kopira sve fajlove sa repozitorijuma i proverava njihovu ispravnost i razlike u samim fajlovima. Kada se proces završi, dobijamo build 1, prvu verziju aplikacije koja se nalazila na repozitorijumu. Završen build bez pojave grešaka možete videti na narednoj slici:



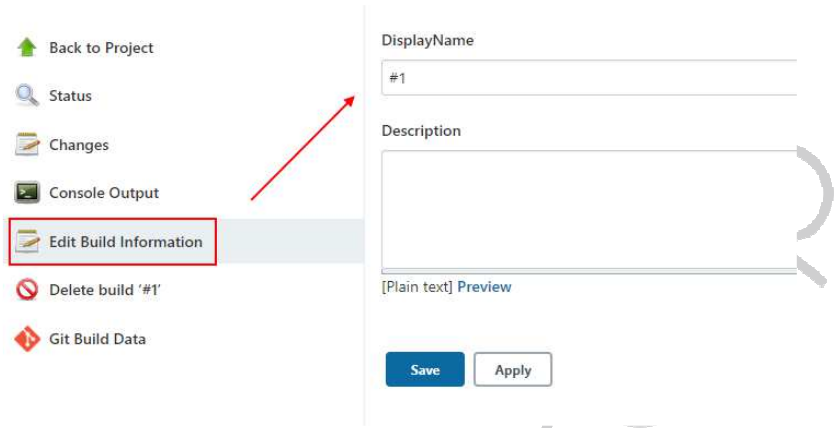
Slika 7.30. Kreiranje prvog builda projekta

Ukoliko sada odaberemo *Workspace* ikonu, dobijamo pregled fajlova koji su se nalazili u repozitorijumu pre kreiranje prve verzije unutar Jenkins projekta. Šta se sada dogodilo? Jenkins je izvršio integraciju fajlova – doduše, ne automatsku, jer smo morali da kliknemo na *Build Now*, ali sada smo dobili sve fajlove repozitorijuma objedinjene na jedno mesto, uključujući i testove koji su postojali na njemu. Dakle, sada smo obavili integraciju našeg projekta.



Slika 7.31. Workspace ili radni folder projekta

Ovo znatno olakšava rad; npr. da imamo još jednog programera i da on doda neki fajl na repozitorijum, izborom *Build now* opcije unutar Jenkinsa, mi bismo dobili i taj novi fajl, kao i izveštaj ko je fajl postavio. Izveštaj možemo videti izborom direktno verzije builda (donji levi ugao ekrana #1). Tada pristupamo opciji same build verzije, pa hajde prvo da promenimo naziv builda, jer nam ovo #1 ne govori puno. Izborom opcije *Edit Build Information* možemo promeniti osnovne podatke o buildu.



Slika 7.32. Promena osnovnih podataka builda

Primera radi, postavićemo naziv na *First Integration 1.0*. Još jedna opcija koja je od velikog značaja za kontrolisanje rada programa i ujedno za praćenje integracije fajlova jeste *Console Output*.

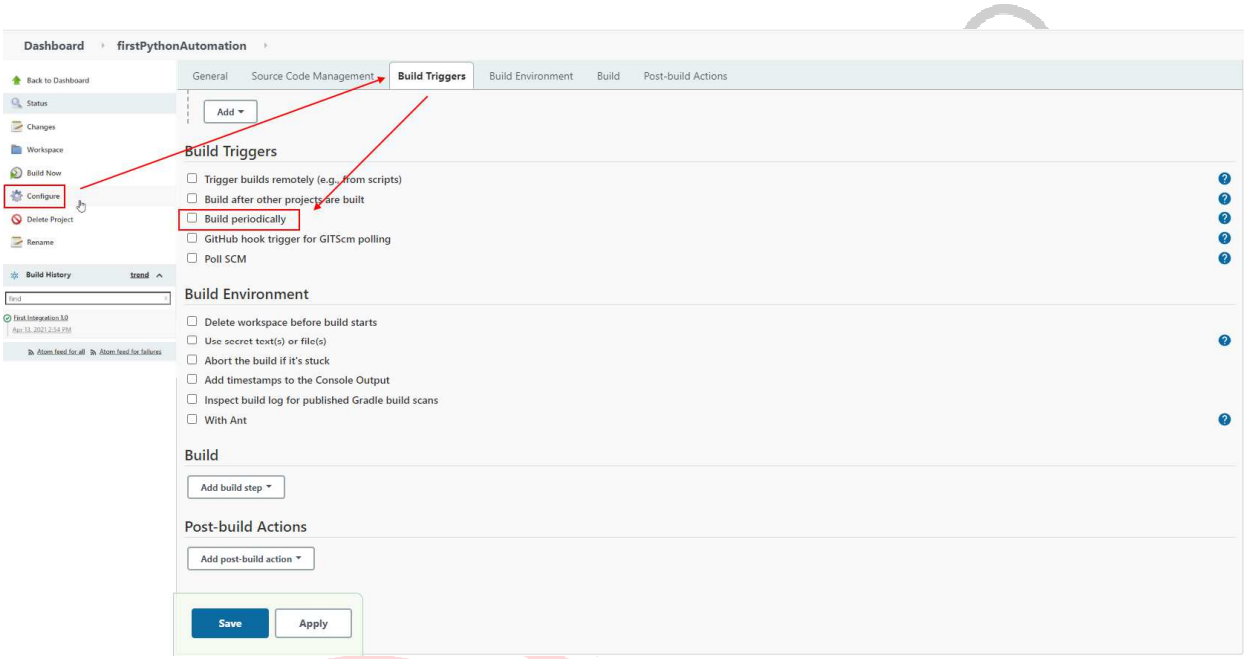
Slika 7.33. Prikaz Console Output za ovaj build

U samoj konzoli možemo videti ko je inicirao spajanje fajlova, sa kojeg repozitorijuma, sa kojim parametrima i kako se sam proces završio. U slučaju grešaka i crvene boje ikonice builda, biće prikazan i razlog zašto je greška nastala.

Automatizacija integracije fajlova

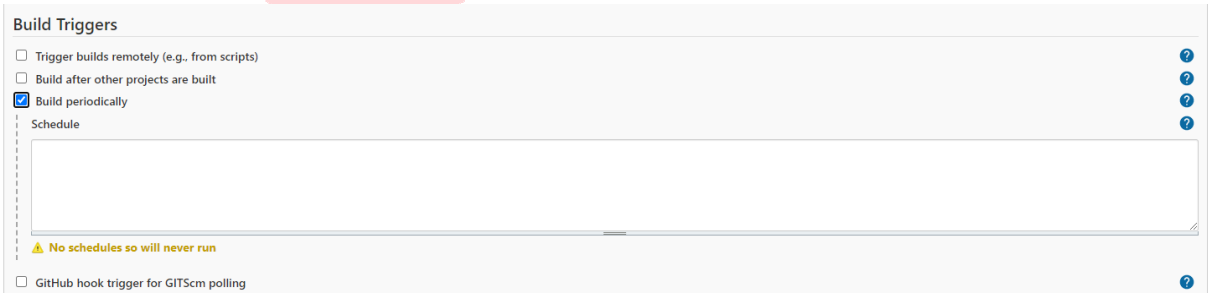
Kao što možete videti, povezivanjem repozitorijuma i izborom opcije *Build now*, izvršili smo integraciju fajlova programa. Ali želimo da postignemo još više: hoćemo da postignemo automatsku integraciju – dakle, da mi ne moramo da kreiramo build, već da se on npr. nekoliko puta u toku dana kreira sam; tada je lakše da krajem dana prođemo kroz izveštaje i uporedimo šta se dogodilo tokom dana, ko je dostavio fajlove i slično.

Ovo takođe možemo lako postići u okviru Jenkinsa, kroz izbor *Configure* opcije našeg projekta, nakon čega u *Build Triggers* sekciji možemo definisati okidače kojima određujemo u kom slučaju želimo da se kreira novi build projekta.



Slika 7.34. Build triggers sekcija

Pored definisanja posebne skripte koja pokreće build proces ili povezivanja sa Git repozitorijumom gde na svaki push repozitorijuma možemo kreirati i build verziju, često korišćena opcija je *Build periodically* (slika 7.34). Izborom ove opcije imamo mogućnost da unesemo vreme ili vremenski interval obavljanja build procesa. Izborom ove opcije dobijamo prikaz kao na narednoj slici.



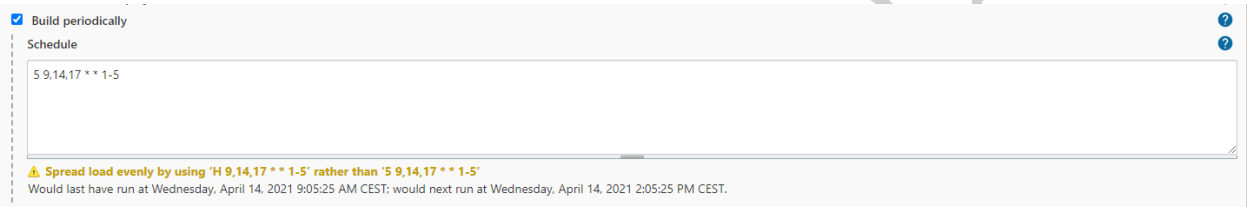
Slika 7.35. Opcija Build periodically

Problem je što ovo polje ne podržava neke od standardnih načina da unesemo vreme ili dane, već je potrebno koristiti kron izraze (cron expressions). Oni predstavljaju poseban vid (sintaksu) zapisivanja vremena i vremenskih intervala. Naravno, za potrebe rada u okviru Jenkinsa i Pythona generalno nemamo preteranu potrebu da uočimo kako tačno zapisivati ove izraze. Na internetu je dostupno na desetine web lokacija gde imamo gotove alate koji nam olakšavaju rad sa kron izrazima. Odličan primer alata ovoga tipa je:

https://crontab.guru/#5_9,14,17_*_*_1-5

Na linku je već podešen izraz koji koristimo za naš projekat, ali promenom parametara možemo dobiti drugo vreme u minutima, satima, danima, mesecima (respektivno).

Dakle, sada izraz: 5 9,14,17 * * 1-5 kopiramo unutar Schedule polja i možemo videti da se prikazuje vremenski raspored kreiranja builda. Nakon toga je dovoljno da odaberete *Apply*, pa *Save* da potvrdite izmenu.



Slika 7.36. Unos kron izraza

Potvrdom ove opcije, definisali smo da se svakog radnog dana u nedelji, u tačno 09:05, 14:05 i 17:05 kreira po jedan build. Na ovaj način smo postigli automatsku integraciju, gde više ne moramo mi uticati na generisanje build verzija, već se prema vremenu u toku dana ona automatski obavlja. Naravno, možemo u svakom trenutku kreirati novi build, korišćenjem opcije *Build now*, kao što smo do sada i radili.

Pitanje

Najbrži proces testiranja programa jeste pomoću:

- continuous integration pristupa
- continuous delivery pristupa
- **continuous deployment pristupa**

Objašnjenje:

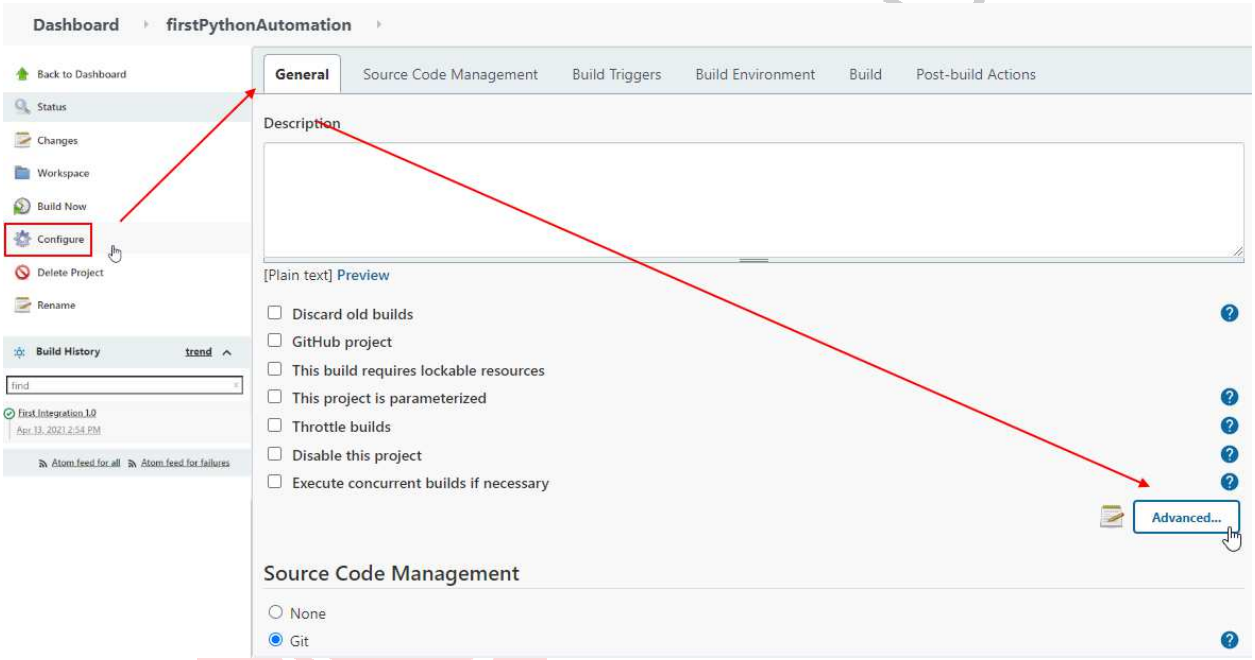
Continuous deployment pristup je toliko automatizovan da nema ljudske intervencije. Svi testovi se obavljaju automatski kako se fajl okači na repozitorijum i ukoliko prođe testove, automatski se objavljuje. Ovo je najbrži vid rada, ali iziskuje ekstremno precizne testove.

Automatizacija testiranja

Naredni korak je da postignemo automatizaciju testiranja. Dakle, da prilikom kreiranja build verzije automatski i pokrenemo određene testove, bilo unit ili integration testove. Na ovaj način, kako smo automatizovali proces kreiranja builda, automatizujemo i izvršavanje potrebnih testova. Upravo ovim postižemo automatizaciju kako integration procesa tako i delivery procesa.

Jenkins ovaj proces znatno olakšava, ali pre svega, napravićemo par podešavanja da bude jasnije i preglednije kako sve funkcioniše.

Pre svega, promenićemo lokaciju Workspace foldera – foldera u kojem Jenkins čuva podatke o projektu. Ovo takođe možemo lako postići u okviru Jenkinsa, tako što izaberemo *Configure* opciju našeg projekta i zatim u *General* sekciji pristupimo *Advanced* opciji, koja nam pruža još dodatnih podešavanja.



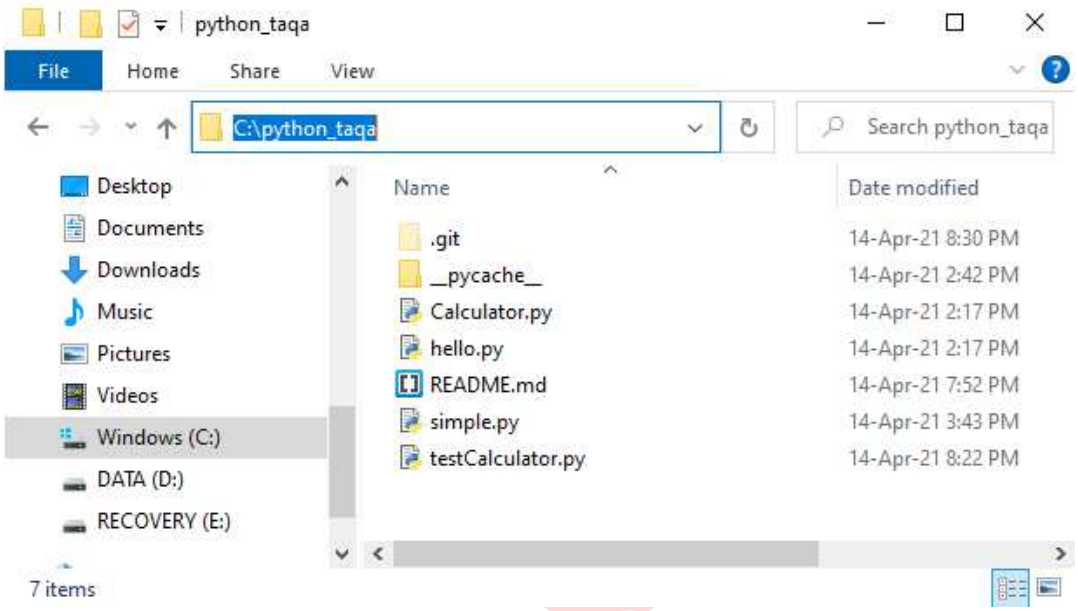
Slika 7.37. Pristup Advanced opcijama projekta

U okviru *Advanced* opcija potrebno je pronaći i čekirati stavku *Use custom workspace*; nakon toga, u *Directory* polja za unos potrebno je navesti tačnu putanju do direktorijuma koji će predstavljati *Workspace* folder. Za ovu svrhu smo na C partriciji našeg diska kreirali folder.

Slika 7.38. Izmena Workspace foldera

Nakon unosa putanje potrebno je povrditi izmene sa *Save*. Sada nam je pristup radnom folderu značajno olakšan, jer podrazumevano Jenkins ovaj folder postavlja duboko unutar sistemskih foldera računara.

Nakon ove izmene, ručno generišemo build verziju sa *Build now* opcijom i u okviru novog definisanog foldera možemo videti fajlove koji su prethodno bili na repozitorijumu.

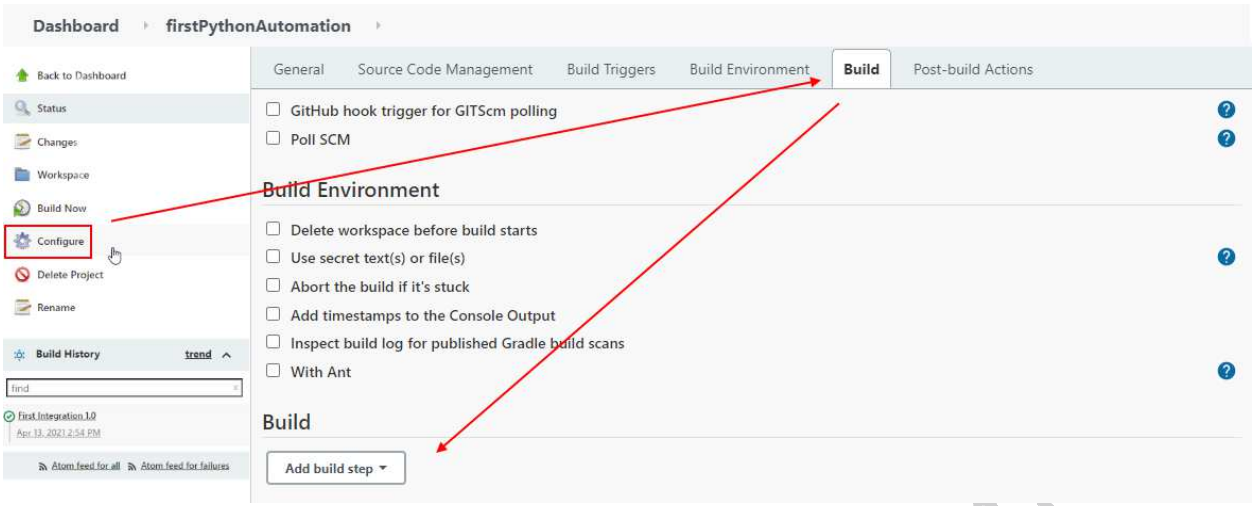


Slika 7.39. Sadržaj novog Workspace foldera nakon builda

Možda se sada pitate zašto nam je značajno to što smo promenili folder. Pre svega, ovo će biti česta praksa, kada želite npr. da napravite ručno novu build verziju, ali ne želite da se prethodni fajlovi izmene. Onda je promena Workspace foldera lak način da napravite novo okruženje u kojem ćete sprovoditi testove. Pored ovoga, u ovaj folder možemo i direktno kopirati fajlove i foldere koji će biti direktno dostupni Jenkins serveru.

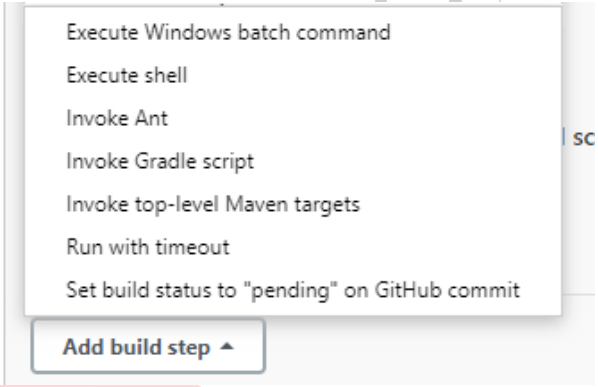
Ali, kao što smo naveli, nama je cilj automatizacija i samog testiranja i što manje naše interakcije sa postupkom spajanja i testiranja aplikacije. Stoga je naredni korak definisanje koraka u procesu generisanja build verzije (build step). Na ovaj način možemo da utičemo na sve procese koji se odvijaju tokom generisanja jedne verzije naše aplikacije.

Pristup build koracima obavljamo takođe pomoću *Configure* opcije našeg projekta; u *Build* sekciji možemo pronaći opciju *Add build step*.



Slika 7.40. Pristup Add build step opciji

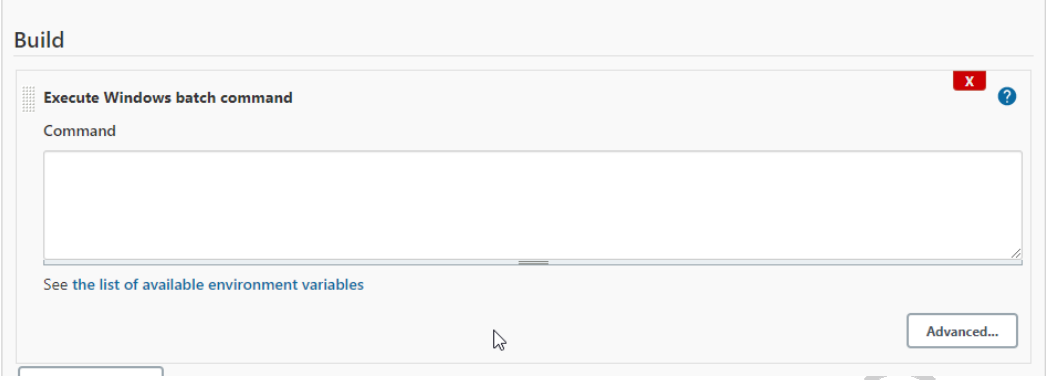
Opcija *Add build step* sadrži veći broj opcija, koje možemo birati u zavisnosti od toga šta je potrebno uraditi tokom procesa generisanja verzije.



Slika 7.41. Opcije Add build step

- *Execute Windows batch command* opcija omogućava izvršavanje komandi i pokretanje fajlova na Windows operativnim sistemima, kao i serverima;
- *Execute Shell* opciju koristimo na Linux operativnim sistemima, kao i na većini današnjih servera;
- *Invoke Ant*, *Gradle* i *Maven* opcije se koriste za druge programske jezike, kao i posebne biblioteke koje se mogu koristiti na projektima;
- *Run with timeout* opcija omogućava da delove builda postavimo na čekanje;
- Na kraju, *Set Build status to "pending"* koristimo kada želimo da promenimo status nekog od commita ka GitHub repozitorijumu.

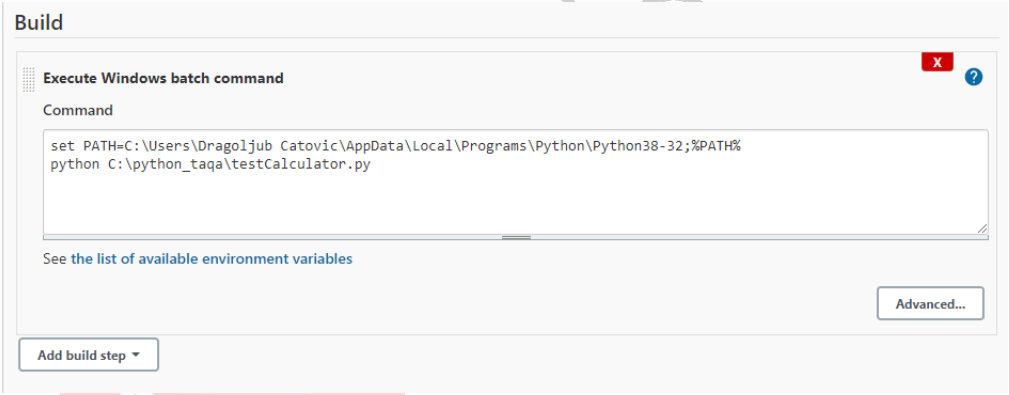
S obzirom na to da koristimo Windows operativni sistem i radimo na našem računaru, potrebna nam je opcija *Execute Windows batch command*. Izborom ove opcije dobijamo prostor za unos komandi, slično kao što bismo imali mogućnost putem *Command Prompta* našeg sistema.



Slika 7.42. Korak Execute Windows batch command

Upravo u ovom polju sada možemo definisati pokretanje testa koji će se izvršiti kako se novi build bude kreirao; za primer ćemo koristiti sledeće linije koda:

```
set PATH=C:\Users\Dragoljub
Catovic\AppData\Local\Programs\Python\Python38-32;%PATH%
python C:\python_taqa\testCalculator.py
```



Slika 7.43. Execute Windows batch command

Sa ove dve linije koda definisali smo, pre svega, na prvoj liniji, tačnu putanju do naše instalacije Pythona, jer je Jenkinsu potrebno da „zna" koji program izvršava kod koji će biti dat u nastavku.

Dakle, linija u zavisnosti od lokacije na vašem računaru vodi ka instalacionom folderu Pythona:

```
set PATH=C:\Users\Dragoljub
Catovic\AppData\Local\Programs\Python\Python38-32;%PATH%
```

A u drugoj liniji koristimo komandu python i dajemo tačnu putanju do fajla koji želimo da pokrenemo.

```
python C:\python_taqa\testCalculator.py
```

Fajl testCalculator.py je predstavljen u prethodnim lekcijama kursa i sadrži jednostavan test integracije između tri funkcije. Ovaj fajl je dostupan na repozitorijumu i stoga je i sinhronizovan u Workspace folderu Jenkinsa.

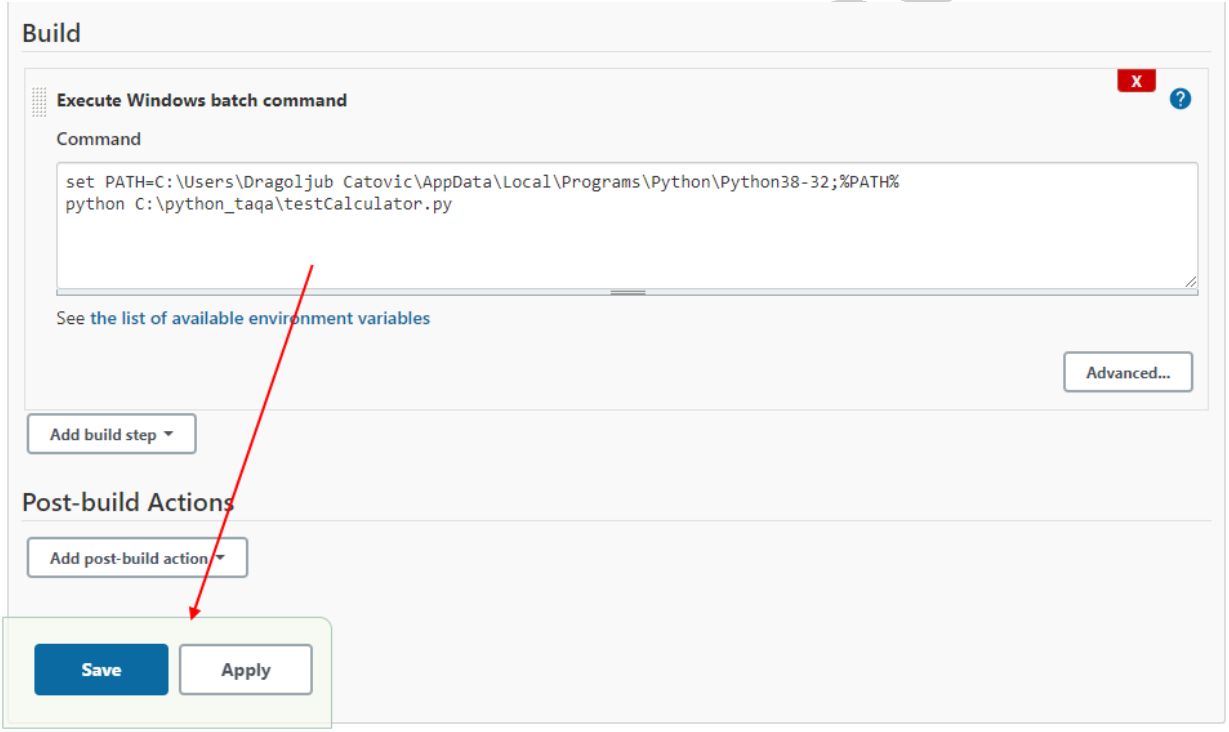
Napomena: Putanja se može razlikovati na vašem računaru u zavisnosti od lokacije i naziva workspace foldera.

Naravno, pored navođenja jednog fajla za pokretanje, ovde mogu postojati desetine fajlova i uvek ih pokrećemo po sintaksi:

```
python path/file.py
```

gde se komanda sa putanjom do fajla dodaje na novoj liniji.

U redu; kada kopiramo kod u polje (slika 7.44), biramo opciju Save i spremni smo za kreiranje novog builda.



Slika 7.44. Dodavanje Windows batch komande

Kada smo kreirali novi build, pristupamo opciji *Console Output*, u okviru novog builda. Prilikom pregleda izveštaja o buildu možemo pri samom dnu uočiti i od ranije poznat izveštaj o izvršenim testovima:

Slika 7.45. Rezultat izvršavanja testa

Dakle, sada smo postigli pre svega to da se build kreira tri puta u toku jednog dana i sa svakim izvršavanjem kopiraju se svi fajlovi sa repozitorijuma. Ovo znači da, bez obzira na to koliko programera doprinosi repozitorijumu, u okviru Jenkinsa možemo videti sve fajlove, kao i promene nad njima. Upravo sa tim smo postigli automatizaciju delivery procesa. Nakon toga smo dodavanjem komandi omogućili da se prilikom automatskog kreiranja builda izvrši i fajl koji u sebi ima integracioni test. Kako se program izvrši, postigli smo i automatsko testiranje na integraciju fajlova. U okviru naredne lekcije govorićemo o poslednjem aspektu automatizacije, a to je automatizacija objavljivanja.

Rezime

- Neprekidna isporuka predstavlja pristup u softverskom inženjerstvu gde se softver proizvodi u kratkim ciklusima, osiguravajući da se delovi softvera mogu objaviti u bilo kom trenutku.
- Cilj neprekidne isporuke jeste da se izrada, testiranje i objavljivanje softvera obavljaju većom brzinom i što češće. Sa ovim pristupom kompanije umanjuju troškove i vreme potrebno za realizaciju i smanjuju rizik od grešaka, jer se aplikacija spaja iz delova automatski.
- Continuous integration (CI) predstavlja pristup u softverskom inženjerstvu gde programeri vrše integraciju koda u deljeni repozitorijum, gde se više puta u toku jednog dana proverava kod i vrši testiranje integracije.
- Continuous delivery (CD) je pristup u kojem programeri razvijaju, verzionišu, testiraju i objavljuju celine programa u veoma kratkim periodima. Ovaj pristup predstavlja nastavak procesa CI. Ovaj pristup se koristi kada se radi update postojećeg softvera, gde je potrebno da se za kratko vreme, a nekada i bez prekida rada programa, izvrši promena na samoj aplikaciji.
- Continuous deployment (CD) je sličan continuous delivery procesu, ali za razliku od prethodno spomenutih, ovaj proces je toliko automatizovan da nema ljudske intervencije. Svi testovi se obavljaju automatski kako se fajl okači na repozitorijum i ukoliko prođe testove, automatski se objavljuje. Ovo je najbrži vid rada, ali iziskuje ekstremno precizne testove.

- Build verzija programa predstavlja verziju programa koji još nije završen. Svaka nova celina programa koja se testira na integraciju će predstavljati jednu verziju, jedan build.
- Release verzija predstavlja verziju programa koja je stabilna i koju korisnik upotrebljava.
- Pojam deploy, u kontekstu software deploymenta, predstavlja celinu koda ili celu aplikaciju koja je postavljena na neko okruženje, gde se testira od strane razvojnog tima, ili alternativno od strane male grupe korisnika, gde se testira rad programa u realnim uslovima.
- Jenkins je vodeći open source server za automatizaciju procesa razvoja softvera. Jenkins je serverska aplikacija i uvek se postavlja na neki vid web servera. Njegova primena na serveru omogućava programu da ostvaruje potrebne konekcije sa drugim udaljenim računarima i serverima za skladištenje podataka.